JUNIA

COO - Projet Médiathèque

F. Chatrie
AP5

1 Introduction

1.1 Objectif du projet

L'objectif du projet est de mettre en œuvre les notions appréhendées durant l'enseignement de ce module. Durant cet enseignement vous avez normalement acquis les notions de base de la conception/programmation orientée objets appliquées au langage C++. Le point de départ du projet est constitué du présent document. Ce dernier correspond au cahier des charges spécifié par le client. Afin de mener à bien ce projet, vous allez devoir :

— lire le cahier des charges,

— identifier les besoins exprimés par le client,

— comprendre ce que vous devez développer,

— élaborer un modèle de solution,

— développer et mettre au point votre application.

Pour mener à bien les quatre premières étapes, il vous est demandé de mettre en pratique les notions d'UML vues en cours. Cela vous permettra de structurer vos idées et de les partager (avec votre binôme et votre enseignant). De plus, étant donné que le projet est un projet de conception orientée objets, il vous est demandé dans les deux dernières étapes de mettre en œuvre les notions vues en cours (héritage, classes, polymorphisme, surcharge, etc.).

2 Déroulement du projet

La durée estimée du projet est d'environ 12 heures. Trois séances de TP de 4 heures sont programmés dans votre emploi du temps.

Le sujet du projet a été écrit de manière à prendre en considération les différences de niveaux entre les groupes. Pour cela le sujet du projet est écrit sous forme de cahier des charges à tiroir. Cette particularité vous permettra d'avancer à votre rythme.

2.1 Évaluation de votre travail

La notation du projet se basera sur :

– La qualité de l'analyse du cahier des charges. Cette analyse doit être réalisée à l'aide d'UML.

– L'utilisation adéquate des possibilités offertes par le langage C++ (héritage, surcharge, ...).

– Le respect des fonctionnalités spécifiées dans le cahier des charges.

L'évaluation du projet sera réalisée en deux parties :

- Lors d'une présentation orale de votre projet.
- Dans un rapport papier détaillant le développement de votre application.

Projet

Lors de la présentation orale de votre application, vous êtes en charge de présenter à votre enseignant (le client) l'application développée. Cette présentation d'une durée approximative de 10 minutes vise à démontrer au client que vous avez développé ce qu'il vous a commandé. Pour cela, vous présenterez vos choix de conception, vous expliquerez les différentes étapes par lesquelles vous êtes passées et enfin réaliserez une démonstration des fonctionnalités de l'outil (démonstration que vous aurez pris soin de préparer).

En ce qui concerne le rapport écrit, une attention particulière sera portée à la présentation des choix d'implémentation que vous aurez réalisés lors de votre développement. Ces choix ainsi que votre cheminement devront être détaillés clairement à l'aide du langage UML.

3 Cahier des charges

Le sujet de projet s'intéresse au développement d'une application permettant de gérer une médiathèque. Les besoins de la médiathèque à laquelle nous allons nous intéresser seront simplifiés par rapport à un cas réel afin de rendre l'étude réalisable dans le temps imparti. Pour cela, nous considérerons par exemple que la bibliothécaire possède des compétences en informatique et que cela ne lui pose pas de difficulté que l'application soit développée en mode console. De plus, dans un premier temps, l'application sera développée en mode mono-utilisateur, cela signifie qu'un seul et unique utilisateur pourra utiliser l'application à un instant donné.

L'application doit permettre de gérer la disponibilité de l'ensemble les ressources disponibles dans la médiathèque : des livres, des CD, des DVD, des revues et des vidéos VHS. Les informations mémorisées pour chacune des ressources contenues dans la médiathèque sont dépendantes du type de la ressource. Les informations liées aux ressources seront par exemple : le titre, l'auteur, la durée, le nombre de pages, etc.

L'application devra au minimum fournir à l'utilisateur les fonctionnalités suivantes :

- Permettre l'ajout et la suppression de médias dans le système de gestion de la médiathèque,
- Permettre la consultation des ressources de la médiathèque (liste des ressources ou informations détaillées relatives à une ressource),
- Permettre de sauvegarder et de charger le contenu de la médiathèque à partir d'un fichier,
- Permettre de rechercher un média à partir d'une information (son titre, le nom de l'auteur, etc.), ces recherches pourront être incrémentales (possibilité de faire une recherche uniquement sur les résultats de la recherche précédente),
- Permettre de réserver, emprunter ou rendre une ressource.

La liste des besoins exprimés ci-dessus n'est pas exhaustive. Vous pourrez l'étoffer en fonction des besoins que vous identifierez (ou bien en fonction de vos envies).

L'application doit permettre de mémoriser toutes les ressources disponibles dans une médiathèque classique. Pour cela nous allons devoir mémoriser les informations issues de différents types de supports : livres, revues, CD, DVD, VHS, etc. L'ensemble de ces différentes ressources sont décrites à l'aide d'informations dépendant de leur type. Une liste non exhaustive des informations relatives aux différents types de ressources est fourni ci-dessous :

- **Des livres** : un livre est écrit par un auteur et publié durant une année précise. Il est composé de N pages et appartient à une certaine collection. De plus un livre possède un titre et il peut posséder un résumé.

- **Des revues** : une revue possède les mêmes caractéristiques que les livres sauf qu'elle possède en plus un éditeur et un ainsi qu'un attribut spécifiant le nombre d'articles contenus dedans. Pour chaque revue on pourra mémoriser le nom des articles contenus afin d'autoriser leur indexation.
- **Des CD** : un CD est un support musical qui possède une durée, un nombre de pistes, un auteur, une maison de production et un titre.
- **Des vidéos VHS** : une VHS est un support vidéo qui possède une durée, un auteur, et une maison de production.
- **Des DVD** : un DVD est un support vidéo similaire à une VHS sauf qu'il possède en plus une information spécifiant le nombre de pistes (ou chapitres) qu'il contient.
- **Des ressources numériques** : une ressource numérique sera pour nous un « simple » fichier. Un fichier possède un auteur, un type (format PDF, DOC, PPT, etc.), une taille ainsi qu'un nom et un chemin d'accès (nous supposerons que ces fichiers sont mémorisés sur un serveur WEB ⇒ le chemin sera donc une URL).

Les interactions entre l'homme et la machine seront réalisées mode ligne de commandes afin de limiter la difficulté. Voici la liste des commandes que devra supporter votre application :

- **BYE** : lorsque l'utilisateur lance cette commande, il demande simplement à quitter votre application. Dans ce cas là, vous devez tout simplement fermer proprement votre outil (dans le sens destruction des ressources allouées).
- **ADD type** : cette commande lancera la procédure permettant de créer une nouvelle ressource en fonction du type spécifié en paramètre.
- **LOAD filename** : cette commande doit permettre à l'utilisateur de charger un fichier contenant les ressources disponibles dans la médiathèque. Le contenu de ce fichier viendra remplacer les données actuellement chargées dans l'outil.
- **SAVE filename** : cette commande permet à l'utilisateur de sauvegarder les données actuellement dans l'outil dans le fichier dont le nom est spécifié en paramètre. Le format de sauvegarde des données est laissé à votre discrétion.
- **SEARCH chaîne** : cette fonction permet de rechercher dans la base de données les documents contenant la chaîne de caractères spécifiée en paramètre. Les résultats issus de la recherche remplaceront temporairement la base de données courante pour faire en sorte que si l'utilisateur lance une seconde recherche elle ne s'appliquera que sur les résultats de la première recherche (recherche incrémentale).
- **CLEAR** : cette commande permet de réinitialiser la base de données courante après une recherche. Si l'utilisateur emploie cette commande après avoir effectué une recherche alors, les résultats sont détruits et la future recherche s'appliquera à l'ensemble des ressources disponibles dans la médiathèque.
- **LIST** : cette fonction permet d'afficher de manière compacte les données contenues dans la base de données. Si une recherche a été lancée, alors cette commande ne doit afficher que les résultats de la recherche précédemment lancée.
- **SHOW id** : cette fonction doit permettre d'afficher les informations détaillées sur une ressource précise contenue dans la médiathèque. Le paramètre passé lors de l'appel de la procédure utilise un identifiant unique que doit posséder toute ressource appartenant à la médiathèque (identifiant de l'objet par exemple ⇒ attribut static).
- **DELETE id** : cette méthode permet de supprimer une ressource particulière appartenant à la médiathèque.
- **RESET** : cette dernière méthode permet de supprimer toutes les ressources contenues dans la bibliothèque.

L'ensemble de ces commandes doit permettre à l'utilisateur de réaliser les tâches décrites dans le cahier des charges. Vous êtes toutefois libre de rajouter de nouvelles fonctionnalités à votre gré.

3.1 Extension des capacités de l'outil

Maintenant que vous avez développé une application capable de gérer une médiathèque nous allons étendre la capacité de l'outil en intégrant la possibilité de gérer plusieurs utilisateurs différents. Pour cela nous allons considérer que nous avons 2 catégories d'utilisateurs : un administrateur et des clients.

Le cahier des charges initial va être modifié de la manière suivante : un client peut consulter les ressources disponibles dans la médiathèque. De plus, un client doit être capable de réaliser les recherches sur les documents présents. L'administrateur possède un accès complet aux fonctions déjà développées dans la partie précédente du projet. Lors du lancement de l'application, l'utilisateur doit fournir le mot de passe administrateur lors d'une phase d'authentification, si l'authentification échoue alors les droits associés à la session courante seront ceux d'un client.

Afin de permettre une véritable utilisation multi-utilisateurs, il est important de pouvoir lancer plusieurs fois l'outil en simultané et que ces derniers travaillent sur la même base d'informations (fichier). Pour que les clients puissent bénéficier des nouveaux produits rajoutés par l'utilisateur, vous devrez rajouter une commande nommée **Reload** et qui permettra de recharger le fichier contenant les données.

En fonction de vos connaissances, vous pourrez développer une méthode capable de recharger automatiquement le fichier lorsque l'administrateur sauvegarde sa session courante. Pour cela vous pouvez par exemple vous baser sur la date d'enregistrement du fichier (qui sera modifiée à chaque enregistrement).

Dans votre rapport vous inclurez les nouveaux diagrammes de cas d'utilisation ainsi que les diagrammes de classes permettant l'implémentation de ces nouvelles fonctionnalités.

3.2 Pour les plus téméraires

Pour ceux qui trouveraient le sujet du projet trop court, il est possible de rajouter à ce dernier une dimension graphique. Cette partie ne doit être considérée que si vous avez terminé proprement la première partie (développement de l'application avec une interface de type terminal).

Afin d'améliorer l'ergonomie de l'outil, on peut souhaiter lui adjoindre une interface graphique. Afin de développer de telles interfaces en C++, il existe la bibliothèque QT de Trolltech qui propose un certain nombre d'objets graphiques facilement utilisables. Vous trouverez des tutoriels pouvant vous aider dans le développement d'une interface graphique pour l'outil.

4 Exemples de codes sources utiles

Le premier exemple de code vous montre comment il est possible de lire l'ensemble d'un fichier de type textuel en C++. Cet exemple lit les informations ligne par ligne et les affiche à l'écran.

```
1 #include <iostream>
2 #include <fstream>
3 #include <string>
4 using namespace std;
5
6 void main()
7 {
8     string STRING;
9     ifstream infile;
```

```
    infile.open("names.txt");
    while(!infile.eof())
    {
        getline(infile, STRING);
        cout << STRING;
    }
    infile.close();
}
```

Dans ce second exemple, le code C++ présenté permet de lire une chaîne de caractère tapée par l'utilisateur dans la console. La lecture de la chaîne de caractères est réalisée lorsque l'utilisateur appuie sur la touche return.

```
void lecture_clavier()
{
    string chaine;
    cout << "Entrez une phrase : " << endl;
    getline(cin, chaine);
    cout << "La phrase est : " << chaine << endl;
}
```

Ce troisième exemple illustre les méthodes disponibles dans la classe string. Ces dernières vous seront sûrement utiles lors de votre développement.

```
void test_string() {
    string c = "Voici une phrase !";
    cout << "Taille de la cha ne      : " << c.length() << endl;
    cout << "Pos. du premier espace   : " << c.find(" ") << endl;
    cout << "Pos. du dernier espace   : " << c.rfind(" ") << endl;
    cout << "Pos. du mot 'une'        : " << c.find("une") << endl;
    cout << "Premier mot              : " << c.substr(0, c.find(" ")) << endl;
    cout << "Comparaison (true)       : " << c.compare(c) << endl;
    cout << "Comparaison (false)      : " << c.compare("TOTO") << endl;
}
```

Ce quatrième exemple vous permet de découper une chaîne de caractères en sous parties en fonction d'un séparateur (ici le caractère « = »).

```
void decoupe_string()
{
    string c;
    cout << "Entrez une phrase : " << endl;
    getline(cin, c);
    int taille = c.size();
    if(taille != 0) {
        int pos = c.find('=');
        cout << "Ligne lue : *" << c << "*" << endl;
        cout << "Position : " << pos << endl;
        cout << "Mot 1 : " << c.substr(0, pos) << endl;
        cout << "Mot 2 : " << c.substr(pos+1, taille-(pos+1)) << endl;
    }
}
```

Ce dernier exemple vous indique la procédure à suivre pour récupérer un pointeur sur la chaîne de caractères contenue dans un objet de type string.

```
void pointeur_char()
{
    string STRING;
    cout << "Votre message = ";
    getline(cin, STRING);
    char* chaine = STRING.c_str();
    cout << "INTEGER = " << atoi(chaine);
}
```